Jaime Fonseca, Carlos Daniel Rebelo, Hélder Costa

Faculdade de engenharia da universidade do porto

Trabalho prático 1

Horários dos estudantes de L.EIC

Introdução

Horários dos estudantes de L.EIC

O trabalho proposto, no âmbito da cadeira de Algoritmos e Estrutura de Dados, visa o desenvolvimento de um sistema capaz de ajudar na gestão de horários após a sua elaboração prévia escrito na linguagem de C++.

Assim, o sistema tem certas funcionalidades que auxiliem nesta gestão, como por exemplo, a listagem de estudantes de acordo com diferentes critérios, a visualização do horário de um estudante em específico, a capacidade de gestão de eventuais pedidos de alteração, entre outros.

A acrescentar a isto, tornar este sistema o mais **eficiente** e **organizado** possível

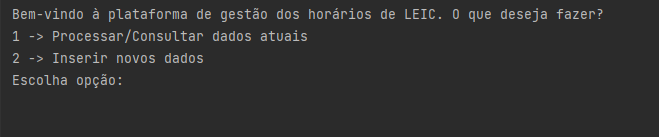
Estrutura do Trabalho

Temos, ao todo, 8 classes definidas (cada uma delas com o seu header e source file) mais a main.cpp. As classes são:

* Student – Contém toda a informação relativa a um estudante específico (nome, studentCode, turmas a que pertence e horário).
* Turma – Contém toda a informação relativa a uma turma em específico (turmaCode, ucCode, estudantes da turma, horário da turma).
* Slot – Definimos slot como sendo a classe que vai conter informação como o dia da semana, horário de início, duração e o tipo relativa a uma “aula”. Por exemplo, uma turma a uma cadeira tem diferentes aulas ao longo da semana, é, portanto, necessário guardar informação sobre cada uma desses períodos.
* Pedido de alteração – Guarda a informação sobre que tipo de pedido foi feito bem como os intervenientes desse pedido. Definimos ao longo do nosso trabalho diferente tipos de pedido, que vão ser explicados mais à frente (adicionar aluno a uma turma, troca de turmas entre 2 dois alunos diferentes ou entre o próprio etc.)
* File Reader – Funciona, como o próprio nome indica, com o objetivo de ler os ficheiros de texto fornecidos e organiza-los em estruturas dados específicos, orgizando-os à medida que o faz através de duas structs criadas.
* Menu – Cria a interface do programa (contem praticamente todos os “couts” utlizados), arquitetada com o objetivo de criar a melhor experiência possível para o utilizador.
* Utils - Contém apenas alguns métodos utilizados ao longo do programa, que não se justificavam por questões de organização e estruturação estarem em outras classes. Métodos como, por exemplo, auxMenu.
* Curso- Funciona como o núcleo do nosso programa que ligará todas estas classes acima descritas, realizando os processos pedidos pelo utilizador.
* Main – Funcionando apenas como um elemento estruturante, mantendo-a com o mínimo de código possível.

Início da Utilização

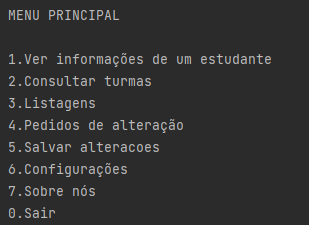
Ao executar o programa pela primeira vez, o utilizador deparar-se-á com a primeira escolha que tem de fazer:



A primeira opção processará os paths predefinidos no programa. No entanto, a diretoria que contém os ficheiros pode, evidentemente, ser alterada de acordo com o computador que esteja a ser utilizado. Por isso é dado ao utilizador a possibilidade de inserir o seu próprio path (absoluto).

Pós-processamento temos 2 estruturas de dados fundamentais do nosso programa. Um set com os estudantes e outro com turmas, organizados através de um operator overload. Utilizamos 2 structs (studentComparator e turmaComparator para o fazer). As estruturas de dados guardam apontadores para os objetos das classes.

Após isso, é apresentado ao utilizador o menu principal:

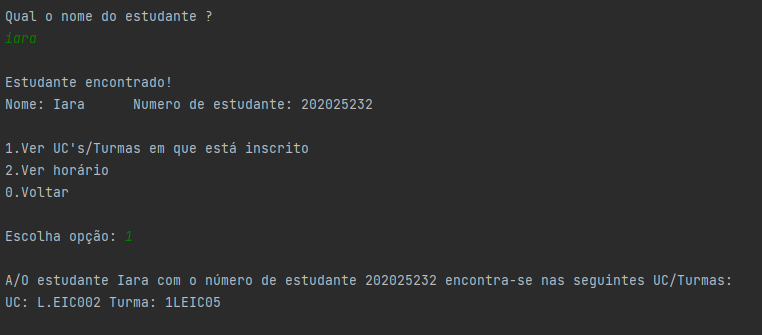


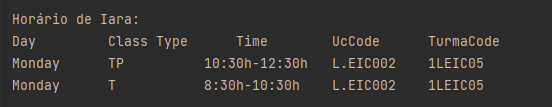
As primeiras 5 opções são autoexplicativas. A opção configurações irá definir à escolha do utilizador aquilo que o mesmo define como sendo o limite, em teoria, máximo de estudantes numa turma (predefinição temos 24). A opção 7 apenas dá print à informação dos envolventes do projeto.

Informações de um Estudante

Ao pretendermos procurar pela informação de um estudante é dada a possibilidade de o pesquisarmos utilizando o nome (não interessando a maneira como escrevemos o nome, letra maiúscula/minúscula) ou pelo seu studentCode através do find\_if de complexidade linear.

Ao encontrarmos um estudante podemos aceder ao vetor turmas com apontadores para todas as turmas que o estudante está inscrito (conseguimos assim consequentemente ver toda a informação sobre cada turma) e o horárioStudent que é criado quando for chamado, que consiste num vetor de pairs que contem a turma e o seu slot. Assim ao dermos print ao horário podemos ver toda a informação necessária sobre cada aula que o aluno vai ter ao longo da semana.

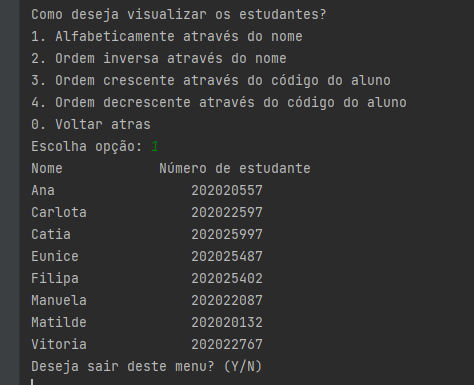
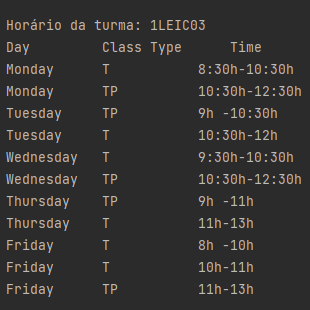




Informações da Turma

Utilizando um método parecido ao do Student apenas adaptado à turma, ao encontrarmos o endereço de memória que contém a turma dada por input, podemos aceder a toda a informação que essa turma contém. O horário da turma (que o utilizador pode especificar se pretende a uma UC ou a todas as UCS) com apenas uma ordem de organização porque considerámos não haver sentido de criar outras ordens de ordenação senão o “normal”, ou os estudantes da turma.

De realçar, a possibilidade de visualizarmos os estudantes de quatro maneiras diferentes como exemplificado em baixo.



Listagens

No menu das listagens, através da utilização de operator overloading para organizar os sets dos estudantes, das funcionalidades da biblioteca STL aplicadas através de lambda functions, são possibilitadas diferentes maneiras de disposição dos estudantes:

* Todos os estudantes inscritos
* Todos os estudantes inscritos em um ano específico
* Todas as turmas
* Número de estudantes inscritos em todas as UC
* Ver estudantes inscritos em certo número de UC
  + Ver estudantes inscritos em mais do que número de UC
  + Ver estudantes inscritos em n UCS
  + Ver estudantes inscritos em menos do que número de UC

Graças à estrutura do programa podemos ver os estudantes inscritos de quatro maneiras diferentes (alfabeticamente, inverso, crescente e decrescente pelo código de estudante)

Pedidos de Alteração

Os pedidos de alteração são feitos utilizando a queue como estrutura de dados para organização. Uma estrutura de dados que à data de realização do trabalho era desconhecida, mas que pareceu a mais adequada dada a natureza de um pedido de alteração. A classe responsabilizada por estes pedidos está organizada de maneira a poder receber diferentes tipos de pedido. Consoante o tipo de pedido efetuado, temos métodos específicos para cada um deles tendo sido esta a maneira que encontrarmos de organizar da melhor maneira possível o código. Os tipos de pedido são:

* Alocar um estudante numa determinada turma/uc
* Remover estudante de uma determina turma/uc
* Troca direta entre dois estudantes de uma turma/uc
* Troca de turma de aluno da mesma Uc

Após a criação do pedido de alteração, o mesmo é então adicionado a uma queue que guarda todos os pedidos de alteração pendentes. No entanto falta, processa-los que envolve a sua validação e execução. Se um pedido for válido por respeitar regras de sobreposição de horário e estudantes inscritos numa turma é então executado. Se não forem validados pelos critérios já mencionados, estas tentativas de pedido são registo no ficheiro de texto archive.

Por fim, na opção de “Salvar alterações” presente no menu principal, será escrito no Student\_classes1 um ficheiro de texto com os pedidos de alteração que foram validados e executados.

Considerações finais sobre o trabalho

Com a realização deste trabalho, podemos dizer seguramente que todos os envolvidos se sentem muito mais à vontade a trabalhar e realizar projetos em C++. O modo de funcionamento das classes e dos seus métodos; uma maior tranquilidade em trabalhar com apontadores, dado que todo o nosso projeto seguiu uma filosofia de trabalho baseada neles, e alertou-nos ainda para eventuais problemas mais específicos tais como as “circular dependencies”. Algoritmos de pesquisa e ordenação com uma nova e basilar preocupação na complexidade de cada linha de código que escrevemos, foram tudo fatores que, seguramente, nos tornaram melhores futuros engenheiros informáticos.